Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

AWK paso a paso... y sin usar el ratón **(47106 lectures)**
Per **Bernardo Cabezas Serra**, ***bernat*** ()
Creado el 20/02/2002 12:22 modificado el 20/02/2002 12:22

*El otro día le consulté una dudilla sobre **awk** a mi jefe, **Javier Peces**. No solo me la resolvió, sino que además dicha duda encendió la mecha que le llevó a escribir este artículo de **introducción al awk** para Bulma. Si es que estos UNIXeros de la vieja escuela están preparados para todo! ;-)*

AWK paso a paso... y sin usar el ratón

Javier Peces(1)

Introducción

Me pide Bernat que le ayude a hacer una faena rápida. Es tan fácil como procesar automáticamente el contenido de una hoja de cálculo. En ella, cada fila contiene datos de un alumno de la Universidad. Se pretende extraer de la hoja el nombre de usuario y la contraseña de cada estudiante, y autorizar su acceso al contenido proporcionado por un servidor web.

Estoy en franca desventaja debido a mis problemas cutáneos (padezco una urticaria que se manifiesta cada vez que me acerco a un ordenador con *Windows*) Un tercer investigador de la Compañía para la que trabajamos se abalanza sobre el problema, debidamente pertrechado con sus cientos de *gigabytes*, gigaciclos y gigaeuros dudosamente gastados en *visualestudios* variados.

Cuando nuestro jovial amigo ha pasado un buen rato buceando por las profundidades del *API* de Excel, extraños problemas relacionados con el flamenco y las Fuerzas Armadas --aparece en la pantalla algo relacionado con *OLE* y el *General Failure*-- le apartan de la cuestión principal. Yo, por mi parte, decido volver sobre el tema realizando una aproximación más humilde.

¿Por qué no *awk*? Al fin y al cabo, esta herramienta fue diseñada en los albores del *UNIX* precisamente para resolver con poco esfuerzo situaciones en las que se desea leer un archivo línea a línea, y tomar decisiones en función del contenido particular de cada registro.

Los adictos a las mil y una variedades del clic (*single click*, *double click*, *right button click*, *close your eyes and clap your hands click*, etc.) encontrarán seguramente alguna ocupación alternativa a la lectura de este artículo. Solamente los locos de la pantalla negra podrán seguir leyendo sin experimentar unas ganas irresistibles de vomitar. El que avisa no es traidor.

Supongamos que el lector selecto --es decir, el que sigue leyendo a pesar de la enorme oferta televisiva del momento-- dispone de un ordenador con su correspondiente sistema operativo y de lo necesario para ejecutar awk. (A decir verdad, *Windows* también puede servir en este caso, ya que existe una versión de *awk* para *msdos*, *win16* , *win32*, etc. Con un pequeño esfuerzo adicional los usuarios de este tipo de *me-reservo-el-calificativo* pueden seguir este artículo a pesar de no tener un sistema decente que llevarse al procesador)

¿Para qué sirve awk?

Como ya se ha indicado, esta utilidad permite procesar secuencialmente uno o varios archivos, los cuales constituyen la información “entrante” para *awk*. El contenido de cada una de las líneas de la entrada se compara con una o varias plantillas, en las que debe especificarse un patrón de búsqueda. A cada patrón se le asocia una acción, que, normalmente, procesa la información entrante y la envía a la salida estándar.

Por tanto, los usos típicos de *awk* se centran en la selección de registros de un archivo en función de uno o varios criterios de búsqueda. Si solamente resultara relevante una parte del registro seleccionado, también sería posible separarla e ignorar el resto.

La información "saliente" se puede transformar según las necesidades del usuario, por lo que es posible crear informes con muy poco esfuerzo de programación.

La sintaxis es muy semejante a la del lenguaje C, con la particularidad de que, al ser *awk* un intérprete, no es necesario pasar por procesos de compilación y *linkedición*. El rendimiento de *awk* no es comparable al de un ejecutable, pero, dada la orientación al proceso secuencial de archivos, esta circunstancia solamente es relevante en condiciones extremas.

Awk puede, además, leer y grabar simultáneamente otros archivos, además de la entrada y salida estándar. Otras capacidades avanzadas quedan, por su mayor dificultad, fuera del alcance de este artículo.

Conviene, finalmente, recalcar la circunstancia de que awk solamente procesa archivos de texto. Para manejar los datos de una hoja Excel o de cualquier otro documento en formato *propietario* es necesario exportar la información a algún formato de texto.

Prerrequisitos

Se presupondrá en adelante que el lector dispone de alguna variedad de *UNIX* (permítaseme un inciso para reconocer el mérito del pedante anónimo que acuñó la expresión *sabores de UNIX*…)

No se requiere el uso de una *shell* concreta. En los ejemplos se utiliza *bash* por ser capaz de interpretar caracteres de control escritos con la sintaxis de escape con “barra invertida”.

Y, evidentemente, se precisa disponer de *awk*. Si se abre una sesión en la consola o una ventana de terminal se averiguará facilmente la presencia de awk en la máquina tecleando la información que aparece en tipografía negrita de paso fijo:

```
$ awk

Usage: awk [-f programfile | 'program'] [-F fieldsep] [-v var=value] [files]
```

Parámetros en tiempo de ejecución

Awk sigue, como hemos indicado, un guión previamente escrito por el programador. Está compuesto por plantillas, que indican --mediante una expresión-- qué registros se desea procesar, y --mediante una acción-- qué proceso se pretende realizar con los registros previamente seleccionados.

Se puede escribir el guión directamente en la línea de mandatos, encerrado entre comillas simples, o en un archivo de texto separado. Si se opta por la segunda opción, debe escribirse el nombre del archivo tras el parámetro –f. La "f" debe ser minúscula obligatoriamente. Por tanto, cualquiera de las dos formas mostradas a continuación produce resultados equivalentes:

```
$ awk `{ print $1 }' entrada.txt

$ awk –f primer.awk entrada.txt
```

Más adelante se mostrará cómo *awk* procesa cada registro como una sucesión de "campos" separados por un carácter determinado que indica dónde termina un campo y empieza el siguiente. La situación más habitual es aquella en la cual cada registro contiene una frase y los campos son las palabras de la frase. El carácter delimitador es el espacio que separa cada palabra de la siguiente. Mediante el parámetro –F (ahora mayúscula) se indica a *awk* qué carácter debe considerar como separador de campos.

```
$ awk –F "#" –f primer.awk entrada.txt
```

También se mostrará la capacidad para usar "variables" dentro del guión, igual que en otros lenguajes de programación. Mediante el parámetro –v se puede asignar un valor a una variable desde la línea de mandatos:

```
$ awk –v fecha="16/Dic/2001" –f primer.awk entrada.txt
```

El resto de los parámetros son los nombres de los archivos de datos "entrantes". El proceso se realiza accediendo a los archivos en el orden especificado. Toma el conjunto de los archivos como si fueran uno solo. *Awk* espera recibir por la entrada estándar (*stdin*) los datos que debe procesar. Por ejemplo:

```
$ cat ejemplo.txt | awk –f primer.awk
```

Alternativamente, los parámetros indicados al final de la línea de mandatos se interpretan como nombres de archivo:

```
$ awk –f primer.awk ejemplo.txt
```

Suponga el lector que se desea actuar sobre un archivo de texto como el que se muestra a continuación. Contiene cuatro líneas de texto terminadas por el carácter de salto de línea "\n", que no se muestra en la pantalla, salvo por el efecto de escribir en la línea siguiente:

```
$ cat ejemplo.txt
La bella y graciosa moza
marchóse a lavar la ropa
la frotó sobre una piedra
y la colgó de un abedul.
$ _
```

Primeros pasos con awk

Todo el proceso gira en torno a la comparación de todas las líneas del archivo, leídas una por una en secuencia, con una o más plantillas. Cada plantilla contiene una expresión, y lleva asociada una acción. La sintaxis es, en general, de la forma:

```
expresión { acción }
```

Si la evaluación de la expresión resulta positiva, se realiza la acción asociada. El resultado de una expresión vacía se considera siempre positivo. Si la acción se omite, se asume que lo que se desea es mostrar el registro en la salida estándar. Este proceso se reitera con cada línea, hasta llegar al final del archivo.

En el siguiente ejemplo se pasa como único parámetro la plantilla entre comillas simples:

```
$ cat ejemplo.txt | awk '1'
La bella y graciosa moza
marchóse a lavar la ropa
la frotó sobre una piedra
y la colgó de un abedul.
$ _
```

Esta plantilla contiene una expresión trivial (1), cuya evaluación resulta siempre positiva. En ausencia de una acción asociada explícitamente, *awk* asume la acción por omisión, que consiste en mostrar la línea completa en *stdout*. Se puede añadir una acción algo más sofisticada entre llaves:

```
$ cat ejemplo.txt | awk '1 { print "línea" }'
línea
línea
línea
línea
$ _
```

Como puede observarse, la acción print se realiza una vez por cada línea que se obtiene de stdin. Para emplear más de una plantilla se emplea la construcción "expresión { acción }" repetidas veces, como en el siguiente ejemplo:

```
$ cat ejemplo.txt | awk  '1 { print "línea" } 1 { print "recta" }'
línea
recta
```

```
línea

recta

línea

recta

línea

recta

$ _
```

También es posible incorporar varias instrucciones en la misma acción, separadas por el signo “punto y coma”:

```
$ cat ejemplo.txt | awk '1 { print “Yo uso”; print “Linux” }'

Yo uso

Linux

Yo uso

Linux

Yo uso

Linux

Yo uso

Linux

$ _
```

Para no confundir al lector, se omite deliberadamente la mención de la posibilidad de usar expresiones regulares semejantes a las que usa *"grep"*. Si el distinguido público lector manifiesta un ferviente deseo de profundizar en el tema, el autor considerará seriamente la posibilidad de ampliar la información al respecto.

Campos de entrada

Para utilizar una expresión que no sea trivial se debe introducir el concepto de “campo”. Se considerará cada registro del archivo de entrada como una sucesión de campos delimitados por un carácter dado. Este carácter es, por omisión, el espacio. Como se ha visto anteriormente, se puede indicar a *awk* que considere otro carácter como separador de campos mediante la opción –F (mayúscula) en la línea de mandatos.

Se hace referencia al contenido de cada campo usando el carácter “$” (dólar) seguido del ordinal del campo: $1, $2, $3, etc. Con la notación $0 se hace referencia a la línea entera.

Se puede forzar el proceso del registro carácter a carácter dejando la variable “separador de campos” FS sin contenido. De este modo, en $1 se tendrá el primer carácter de la línea, en $2 el segundo, etc. Incluso en este caso, el campo $0 contendrá la línea completa.

Como quiera que se está anticipando en estos ejemplos el uso de la instrucción *"print"*, se muestra a continuación el guión anterior con las modificaciones necesarias para mostrar en stdout el contenido de la primera y la segunda palabra

de cada línea:

```
$ cat ejemplo.txt \
 | awk '1 { print "Primera y segunda palabras ===>", $1, $2, "<=== final del registro." }'

Primera y segunda palabras ===> La bella <=== final del registro.

Primera y segunda palabras ===> marchóse a <=== final del registro.

Primera y segunda palabras ===> la frotó <=== final del registro.

Primera y segunda palabras ===> y la <=== final del registro.

$ _
```

Se estudiará más adelante el efecto de la coma en la instrucción "*print*". Si no se indica otra cosa, *awk* inserta un espacio por cada coma que aparezca en esta instrucción.

Variables

También es necesario en este punto citar la existencia de variables predefinidas en el lenguaje. El concepto tradicional de "variable" (almacén de datos en memoria, caracterizado por un nombre no repetido en el mismo contexto y capaz de contener un dato que puede ser modificado por el programa) se aplica perfectamente en este caso.

El programador puede crear sus propias variables símplemente haciendo referencia a ellas en expresiones. Las variables pueden ser escalares (con un solo nombre y valor) o vectoriales (con nombre, subíndice y valor) El subíndice puede ser una cadena arbitraria, lo cual permite crear tablas asociativas. Se emplea la notación "nombre[subíndice]" para referirse a un elemento de la tabla. También se admite la definición de variables de naturaleza matricial, con nombre, varios subíndices y varios valores, si se usa la notación "nombre[subíndice1, subíndice2, subíndice3]".

Las siguientes variables predefinidas están relacionadas con la información entrante:

FS (Field separator): contiene el carácter que indica a awk en qué punto del registro acaba un campo y empieza el siguiente. Por omisión es un espacio. Se puede indicar un carácter alternativo mediante una instrucción de asignación como FS = "/". Si se deja vacío, cada lectura se realiza dejando un carácter en cada campo.

NF (Number of fields): contiene el número total de campos que contiene el registro que se está leyendo en cada momento.

RS (Record separator): contiene el carácter que indica a awk en qué punto del archivo acaba un registro y empieza el siguiente. Es "\n" por omisión.

NR (Number of record): contiene el número de orden del registro que se está procesando en cada momento.

Del mismo modo, se dispone de variables relativas a la información de salida, como las siguientes:

OFS (Output FS): contiene el separador de campos para la salida generada por awk. La instrucción print inserta en la salida un carácter de separación cada vez que aparece una coma en el código. Por ejemplo:

```
$ echo "manejo un apple" | awk -v OFS="\t" `1 {print $1, $2, $3}'

manejo    un      apple

$ _
```

ORS (Output RS): Contiene el carácter que awk escribirá al final de cada registro. Es "\n" por omisión. Obsérvese que en esta modificación del ejemplo anterior la entrada contiene DOS registros, dada la aparición del salto de línea en el centro de la cadena de caracteres entre comillas. Al modificar el carácter separador de la salida se genera un solo registro:

```
$ echo –e "manejo un apple\ncaro y elegante" \
 | awk –v OFS = "\t" –v ORS = "\t"  '1 {print $1, $2, $3}'

manejo    un      apple    caro    y    elegante

$ _
```

Operadores

Son aplicables los operadores comunes en C para realizar operaciones aritméticas sobre variables con valores numéricos, y para concatenar los valores contenidos en variables con valores alfanuméricos. Así, se tiene:

+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto
^	Exponenciación
Esp	Concatenación

También se dispone de los operadores comunes

!	Negación
A ++	Incrementar positivamente en una unidad
A --	Incrementar negativamente en una unidad
A += N	Incrementar positivamente en N unidades
A -= N	Incrementar negativamente en N unidades
A *= N	Multiplicar por N
A /= N	Dividir por N
A %= N	Reemplazar por el resto de la división por N
A ^= N	Sinceramente, no sé qué demonios hace este operador
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

==	Igual a
!=	Distinto de
?:	Formato alternativo de decisión (if/else)

Awk en acción

Para recapitular lo visto hasta ahora, se realizará a continuación algún proceso sencillo con el archivo de ejemplo mostrado en párrafos anteriores. El archivo entrante contiene un número teóricamente no determinado de registros, terminados por el carácter contenido en RS. El proceso comienza leyendo el primer registro, aplicando a éste todas las plantillas existentes con el valor de la variable NR igual a 1. Se reitera el proceso para cada registro, incrementando en una unidad el valor de NR cada vez.

Cada registro está compuesto por NF campos, delimitados por las apariciones del carácter FS en el mismo. El ordinal de cada campo se designa, como se indicó arriba, con $1, $2, $3… y así hasta $NF.

Con campos, variables y operadores se puede escribir una plantilla algo más útil que las anteriores:

```
$ awk '$5 == "moza" { print "La linea", NR, "contiene MOZA en el quinto campo"; }' ejemplo.txt

La línea 1 contiene MOZA en el quinto campo

$ _
```

A medida que la lógica de las plantillas se complique, será de utilidad el parámetro –f (minúscula) seguido del nombre del guión o script que contendrá las plantillas. Cuando las plantillas se codifican dentro del script, no se usan las comillas simples para delimitarlas. Un guión típico como el que se muestra ("proceso.awk" en el directorio actual) podría tener el siguiente contenido:

```
$ cat proceso.awk

{ print "Procesando la línea", NR }

NF > 1 { print "La línea tiene más de una palabra" }

$1 == "Linux" {print "La primera palabra es Linux"}

$ _
```

La primera plantilla siempre se ejecuta por no tener expresión delante de las llaves. La segunda solamente se activa para las líneas que tienen más de una palabra. La tercera es aplicable si la línea comienza con la palabra "Linux".

Plantillas de prólogo y epílogo

Dos expresiones especiales, BEGIN y END, permiten realizar acciones antes de procesar el primer registro, y después de procesar el último, respectivamente.

El sencillo script "suma.awk" pone a cero la variable "total" antes de iniciar el proceso del archivo de entrada, añade a

esa variable el valor correspondiente al contenido del primer campo de cada registro, y muestra el valor de la variable al final:

```
$ cat suma.awk

BEGIN { total = 0 }

{ total += $1 }

END { print "El total es", total }
```

Para ejecutarlo, se puede usar lo siguiente (obsérvese que la entrada tiene cuatro registros):

```
$ echo -e "10\n10\n10\n70" | awk -f ./suma.awk

El total es 100
```

Dependiendo de la implementación de UNIX que se esté usando, el script puede fallar si el campo que interviene en la operación ($1 en este caso) contiene un valor no numérico. En este caso se puede forzar la conversión sumando "cero" a la variable.

Instrucciones de control

Algunas instrucciones del lenguaje C están presentes en *awk* de manera simplificada. Se dispone de la construcción condicional *"if/else"* , que se codifica de la manera habitual:

```
if( expression ) statement [ else statement ]
```

En algunas implementaciones se puede complicar hasta añadir varias instrucciones mediante anidamiento de acciones:

```
{

     print "Procesando registro", NR;

 if( $1 > 0 )

 {
  print "positivo";

  print "El primer campo tiene:", $1;

 }

 else

 {

  print "negativo";

 }

}
```

La primera forma de la construcción *"for"* se maneja igual que en el lenguaje C, es decir:

```
for( expression; expression; expression ) statement
```

donde la primera expresión suele poner valor inicial al índice. En la segunda se controla hasta qué momento la ejecución debe continuar. La tercera marca el nivel de incremento de la variable índice.

```
{
 for( i = 0; i < 10; i ++ )
 {
  j = 2 * i;
  print i, j;
 }
}
```

En su segunda forma, la construcción *"for"* recorre los elementos de una matriz *(array)*. Dado el carácter avanzado de este concepto, quedará para un segundo artículo su manejo.

La construcción *"do/while"* se emplea en la forma habitual. En el caso actual se realiza un sencillo contador de cero a diez:

```
BEGIN { i = 0; do {print i; i ++} while ( i < 10 ) }
```

La instrucción *"break"* permite abandonar un bucle anticipadamente. Para descartar el resto del código del bucle y empezar una nueva iteración se usa *"continue"*. La instrucción *"next"* descarta todas las plantillas que siguen, y pasa a realizar el proceso del siguiente registro. Finalmente, la instrucción *"nextfile"* termina anticipadamente el proceso de un archivo de entrada para pasar al siguiente.

Funciones

Las funciones predefinidas del lenguaje responden al concepto habitual en otros lenguajes de programación. La llamada a una función se resuelve en tiempo de ejecución. Cuando se evalúa una expresión, se realiza la sustitución de la llamada a la función por el valor que ésta devuelve. Por ejemplo:

```
$ awk 'BEGIN { print "El seno de 30 grados es", sin( 3.141592653589 / 6 ); }' /dev/null
El seno de 30 grados es 0.5
$ _
```

Se dispone de las funciones matemáticas y trigonométricas exp, log, sqrt, sin, cos y atan2. Además, son de mucha utilidad las siguientes:

length( )	Longitud del parámetro en bytes
rand( )	Número al azar entre cero y uno
srand( )	Inicia la semilla de generación al azar
int( )	Devuelve el parámetro convertido en un entero
substr(s,m,n)	Devuelve la subcadena de s en m con longitud N
index(s,t)	Posición de s donde aparece t, o cero si no está.
match(s,r)	Posición de s en que se cumple la expresión r.
split(s,a,fs)	Devuelve s en elementos separados por fs
sub(r,t,s)	Cambia en s la cadena t por r. Es $0 por omisión
gsub(r,t,s)	Igual, pero cambia todas las t en la cadena
sprintf(f,e,e…)	Devuelve cadena de formato f con las “e”
system(cmd)	Ejecuta cmd y devuelve el código de retorno
tolower(s)	Devuelve s convertida en minúsculas
toupper(s)	Devuelve s convertida en mayúsculas
getline	Fuerza una lectura de fichero

La solución definitiva

El lector avezado debería estar, a estas alturas, preparado para afrontar el desafío del principio del artículo. Con los mimbres mostrados en las páginas anteriores se tratará de hacer el cesto deseado. Será suficiente con que el guión *awk* resultante sea capaz de construir un *shell script* que contenga los mandatos necesarios para autorizar el acceso de los alumnos al servidor web.

La información de partida está contenida en el archivo “alumnos.txt”, que contiene una línea por cada alumno. En esa línea encontraremos el nombre, los apellidos, la cuenta de usuario que le corresponde y la contraseña que va a emplear para acceder.

```
$ cat alumnos.txt

Sebastián Palomo Linares        spl18901    jalisco

Pedro Pérez Pino             ppp0        hatalue

John Jensen                jotajota    whatafew

María del Carmen García Feo    mc98819    uglymom

$ _
```

En la salida se desea generar un *shell script* con el comentario que indica el nombre de la shell a emplear (*“#!/bin/bash”*, por ejemplo) en la primera línea, y con una línea como *“/usr/bin/htpasswd –b usuario contraseña"* por cada alumno.

Para poner una pizca de dificultad al problema, pondremos de manifiesto un hecho que seguramente no extrañará al lector; los nombres no tienen un número fijo de palabras, y no se garantiza que todos los alumnos estén expresados con los dos apellidos. Así que el planteamiento simplista que haría un anglosajón no muy despierto, consistente en asociar

$1 con el nombre, $2 con el apellido, $3 con la cuenta y $4 con la contraseña, no conduce al resultado correcto:

```
$ awk '{print "/usr/bin/htpasswd -b", $3, $4}' alumnos.txt
/usr/bin/htpasswd -b Linares spl18901
/usr/bin/htpasswd -b Pino ppp0
/usr/bin/htpasswd -b jotajota whatafew
/usr/bin/htpasswd -b Carmen García
```

Otra dificultad que puede presentarse viene dada por la eventual inexistencia de la base de datos de autorizaciones del servidor *web* en el momento de la ejecución del *script*. A los efectos de la programación, implica añadir la opción "-c" a la primera llamada al mandato *"htpasswd"* dentro del script que se genera.

La aproximación correcta al problema consiste en obtener el número de campos (palabras) del registro (línea), y seleccionar la penúltima y la última. Afortunadamente, la variable NF contiene en cada momento el número de campos de la línea que se está procesando. En este ejemplo puede verse cómo se hace uso de esta capacidad:

```
$ awk '{ print NF }' alumnos.txt
5
5
4
7
```

Pero, lo que es realmente maravilloso en este caso es que podemos acceder al valor del campo que se encuentra en el último lugar con tan solo $NF en lugar de NF:

```
$ awk '{ print NF, ":", $NF }' alumnos.txt
5 : jalisco
5 : hatalue
4 : whatafew
7 : uglymom
```

Rizando más el rizo, se puede acceder también al penúltimo campo empleando una expresión aritmética:

```
$ awk '{ print $(NF - 1) }' alumnos.txt
spl18901
ppp0
jotajota
mc98819
```

Combinando ambos, es fácil obtener cuenta y contraseña con la instrucción:

```
$ awk '{ print $(NF - 1), $NF }' alumnos.txt
spl18901 jalisco
ppp0 hatalue
jotajota whatafew
mc98819 uglymom
$ _
```

Para dar los toques finales con más comodidad se escribira el guión en un archivo llamado “altaweb.awk” :

```
$ cat altaweb.awk
BEGIN { print "#!/bin/bash" }
{
    print "/usr/bin/htpasswd",
        ( NR == 1 ? "-c -b" : "-b" ),
        $(NF - 1), $NF
}
END { print "# Shell script generado OK." }
$ _
```

Puede observarse que se ha usado la sintaxis abreviada de la estructura de control *“if/else”*, típica del lenguaje C, en la que una expresión antecede al signo de interrogación. Al evaluarse positivamente se toma en cuenta el código que aparece antes del signo de dos puntos. Si la evaluación es negativa, el código que se ejecuta es el que sigue a los dos puntos.

La ejecución del proceso se realiza, como antes, con el mandato que se muestra a continuación:

```
$ awk -f altaweb.awk alumnos.txt > altaweb.sh
```

Solamente resta verificar el contenido del *shell script* y ejecutarlo. Cuando se tenga confianza en el resultado se puede obviar el paso intermedio y ejecutar directamente:

```
$ awk -f altaweb.awk alumnos.txt | sh
```

Conclusión

Awk es un poderoso intérprete, que permite realizar tareas con archivos de tipo secuencial con una elevada efectividad. La eficacia de su uso en procesos automatizados es innegable, a la vista de que mi jovial colaborador ha desistido, y se dispone a reinstalar *Windows*. Sin embargo, disfrutará de lo lindo con su salvapantallas de tiburones simultaneado con apariciones esporádicas de una misteriosa pantalla azul en la que se glosan las maravillas de la *"Excepción 0E"*…
Hasta más ver.

Bibliografía

1. man awk

2. A.V. Aho, P. J. Weinberger y B. W. Kernighan, *"The AWK programming language"*, Addison-Wesley, 1988. ISBN 0-201-07981-X.

3. Página de Brian Kernighan en Bell Labs, http://www.cs.bell-labs.com/who/bwk/index.html

---

Lista de enlaces de este artículo:

1. mailto:fjpecesARROBAtelelinePUNTOes

E-mail del autor: bercab _ARROBA_ teleline.es
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1201